**Day 3 - 21st  May 2025**

**Task 1:**

Recap of Last session:

Create a file names 21st May.txt and push it to your git hub.  
DONE

**Task 2:**

1. **ACID properties** (Atomicity, Consistency, Isolation, Durability)

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These are four crucial properties that guarantee the reliability and integrity of transactions in a database system.

* **Atomicity**: This means "all or nothing." A set of database operations either completes entirely, with all its changes saved, or it completely fails, and none of its changes are saved. There's no in-between state.
* **Consistency**: This ensures that a transaction brings the database from one valid state to another. It means that any data written must follow all defined rules and constraints, maintaining the database's integrity.
* **Isolation**: This guarantees that multiple transactions happening at the same time don't interfere with each other. Each transaction runs as if it were the only one happening, preventing concurrent operations from corrupting data.
* **Durability**: This ensures that once a transaction is successfully completed (committed), its changes are permanent and will survive any future system failures, like power outages or crashes. The data is safely recorded and will be there when the system recovers.

1. what is scalable, replica set, Scalable, Resilient, and Mission Critical
2. Authorization and authentication, auditing, encryption

Okay, let's go through these introductions. I'll aim for clarity and conciseness for each.

**1. Introduction to Cloud**

**Cloud computing** is the delivery of on-demand computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet1 ("the cloud"). Instead of2 owning and maintaining your own computing infrastructure, you access these services from a third-party provider (like AWS, Azure, Google Cloud) and pay only for what you use.

**2. Introduction to Jenkins**

**Jenkins** is an open-source automation server that helps automate parts of the software development process, particularly for **Continuous Integration (CI)** and **Continuous Delivery (CD)**. It automatically builds, tests, and deploys code changes whenever they are committed, helping developers find bugs faster and release software more frequently.

**3. Introduction to Chef**

**Chef** is an automation platform used for **infrastructure as code (IaC)**. It allows you to define your IT infrastructure (servers, applications, configurations) as code, meaning you write specifications for how systems should be set up and managed. Chef then automatically enforces this desired state across your infrastructure, ensuring consistency and reducing manual configuration errors.

**4. Introduction to Containers**

**Containers** are lightweight, standalone, executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries, and settings. They3 virtualize the operating system, ensuring an application runs consistently across any environment, from a developer's laptop to production servers.

**5. Introduction to DevOps — CI/CD, Meaning, Principles & Examples**

* **Meaning:** **DevOps** is a set of practices, cultural philosophies, and tools that integrate software development (Dev) and IT operations (Ops) teams. The goal is to shorten the systems development life cycle and provide continuous delivery with high software quality.
* **Principles:** Key principles include automation, continuous integration, continuous delivery, collaboration, feedback loops, and continuous improvement.
* **CI/CD:**
  + **Continuous Integration (CI):** Developers frequently merge their code changes into a central repository, after which automated builds and4 tests are run. This helps detect integration errors early.
  + **Continuous Delivery (CD):** An extension of CI, where code changes are automatically built, tested, and prepared for release to production. This ensures that the software can be released reliably at any time.
  + **Example:** A developer commits code, Jenkins (CI) automatically builds it and runs tests. If tests pass, the code is automatically packaged and ready for deployment (CD).

**6. DevOps Lifecycle — Different Phases Explained with Examples**

The DevOps lifecycle is continuous, often visualized as an infinite loop:

* **Plan:** Define goals, requirements, and project scope. (e.g., *Planning features for the next mobile app update.*)
* **Code:** Developers write code and commit changes to a version control system. (e.g., *Writing new code for user login functionality.*)
* **Build:** Compile code, run automated tests, and package the application. (e.g., *Jenkins automatically compiling the code and running unit tests.*)
* **Test:** Perform various types of testing (integration, system, performance) to ensure quality. (e.g., *Automated tests running in a staging environment to catch bugs.*)
* **Release:** Prepare and approve the application for deployment. (e.g., *Gaining approval to push the new features to production.*)
* **Deploy:** Automatically or manually deploy the application to production environments. (e.g., *Automatically pushing the application to cloud servers.*)
* **Operate:** Manage and monitor the application in production. (e.g., *Monitoring server health and application performance 24/7.*)
* **Monitor:** Collect data on application performance and user experience to identify issues or areas for improvement, feeding back into planning. (e.g., *Collecting logs and metrics to analyze user behavior and identify bottlenecks.*)

**7. Introduction to Docker**

**Docker** is the most popular platform for building, shipping, and running applications using **containers**. It provides the tools and framework to easily create, deploy, and manage these portable, self-sufficient application packages. It essentially simplifies the process of containerization.

**8. Purpose of using Docker**

The primary purposes of using Docker are:

* **Consistency:** Ensures applications run the same way everywhere (developer machine, testing, production), eliminating "it works on my machine" problems.
* **Isolation:** Applications and their dependencies are isolated from each other and the host system, preventing conflicts.
* **Portability:** Containers can be easily moved and run across different environments (laptops, on-prem servers, cloud).
* **Efficiency:** Containers are lightweight and start quickly, making better use of resources compared to traditional virtual machines.
* **Faster Deployment:** Standardized packaging and isolation lead to quicker and more reliable deployments.